# **mySQL执行计划**

## **语法**

****explain**** ****<sql语句>****

例如:

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. explain select \* from t3 where id=3952602;

explain输出解释  
   
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+  
| id | select\_type | table | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+

### **id**

SQL执行的顺利的标识,SQL从大到小的执行.  
例如:  
mysql> explain select \* from (select \* from ( select \* from t3 where id=3952602) a) b;  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
| id | select\_type | table      | type   | possible\_keys     | key     | key\_len | ref  | rows | Extra |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
|  1 | PRIMARY     | <derived2> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  2 | DERIVED     | <derived3> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  3 | DERIVED     | t3         | const  | PRIMARY,idx\_t3\_id | PRIMARY | 4       |      |    1 |       |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
 很显然这条SQL是从里向外的执行,就是从id=3 向上执行.

### **select\_type**

就是select类型,可以有以下几种

****SIMPLE****

简单SELECT(不使用UNION或子查询等) 例如:  
mysql> explain select \* from t3 where id=3952602;  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+  
| id | select\_type | table | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+  
|  1 | SIMPLE      | t3    | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 |       |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+

****PRIMARY****

  我的理解是最外层的select.例如:  
  mysql> explain select \* from (select \* from t3 where id=3952602) a ;  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
| id | select\_type | table      | type   | possible\_keys     | key     | key\_len | ref  | rows | Extra |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
|  1 | PRIMARY     | <derived2> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  2 | DERIVED     | t3         | const  | PRIMARY,idx\_t3\_id | PRIMARY | 4       |      |    1 |       |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+

****UNION****

  UNION中的第二个或后面的SELECT语句.例如  
mysql> explain select \* from t3 where id=3952602 union all select \* from t3 ;  
+----+--------------+------------+-------+-------------------+---------+---------+-------+------+-------+  
| id | select\_type  | table      | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra |  
+----+--------------+------------+-------+-------------------+---------+---------+-------+------+-------+  
|  1 | PRIMARY      | t3         | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 |       |  
|  2 | UNION        | t3         | ALL   | NULL              | NULL    | NULL    | NULL  | 1000 |       |  
|NULL | UNION RESULT | <union1,2> | ALL   | NULL              | NULL    | NULL    | NULL  | NULL |       |  
+----+--------------+------------+-------+-------------------+---------+---------+-------+------+-------+

****DEPENDENT UNION****

  UNION中的第二个或后面的SELECT语句，取决于外面的查询  
mysql> explain select \* from t3 where id in (select id from t3 where id=3952602 union all select id from t3)  ;  
+----+--------------------+------------+--------+-------------------+---------+---------+-------+------+--------------------------+  
| id | select\_type        | table      | type   | possible\_keys     | key     | key\_len | ref   | rows | Extra                    |  
+----+--------------------+------------+--------+-------------------+---------+---------+-------+------+--------------------------+  
|  1 | PRIMARY            | t3         | ALL    | NULL              | NULL    | NULL    | NULL  | 1000 | Using where              |  
|  2 | DEPENDENT SUBQUERY | t3         | const  | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 | Using index              |  
|  3 | DEPENDENT UNION    | t3         | eq\_ref | PRIMARY,idx\_t3\_id | PRIMARY | 4       | func  |    1 | Using where; Using index |  
|NULL | UNION RESULT       | <union2,3> | ALL    | NULL              | NULL    | NULL    | NULL  | NULL |                          |  
+----+--------------------+------------+--------+-------------------+---------+---------+-------+------+--------------------------+

****UNION RESULT****

  UNION的结果。  
  mysql> explain select \* from t3 where id=3952602 union all select \* from t3 ;  
+----+--------------+------------+-------+-------------------+---------+---------+-------+------+-------+  
| id | select\_type  | table      | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra |  
+----+--------------+------------+-------+-------------------+---------+---------+-------+------+-------+  
|  1 | PRIMARY      | t3         | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 |       |  
|  2 | UNION        | t3         | ALL   | NULL              | NULL    | NULL    | NULL  | 1000 |       |  
|NULL | UNION RESULT | <union1,2> | ALL   | NULL              | NULL    | NULL    | NULL  | NULL |       |  
+----+--------------+------------+-------+-------------------+---------+---------+-------+------+-------+

****SUBQUERY****

  子查询中的第一个SELECT.  
  mysql> explain select \* from t3 where id = (select id from t3 where id=3952602 )  ;  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------------+  
| id | select\_type | table | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra       |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------------+  
|  1 | PRIMARY     | t3    | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 |             |  
|  2 | SUBQUERY    | t3    | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       |       |    1 | Using index |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------------+

****DEPENDENT SUBQUERY****

  子查询中的第一个SELECT，取决于外面的查询  
  mysql> explain select id from t3 where id in (select id from t3 where id=3952602 )  ;  
+----+--------------------+-------+-------+-------------------+---------+---------+-------+------+--------------------------+  
| id | select\_type        | table | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra                    |  
+----+--------------------+-------+-------+-------------------+---------+---------+-------+------+--------------------------+  
|  1 | PRIMARY            | t3    | index | NULL              | PRIMARY | 4       | NULL  | 1000 | Using where; Using index |  
|  2 | DEPENDENT SUBQUERY | t3    | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 | Using index              |  
+----+--------------------+-------+-------+-------------------+---------+---------+-------+------+--------------------------+  
   
   
****DERIVED****

  派生表的SELECT(FROM子句的子查询)  
  mysql> explain select \* from (select \* from t3 where id=3952602) a ;  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
| id | select\_type | table      | type   | possible\_keys     | key     | key\_len | ref  | rows | Extra |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
|  1 | PRIMARY     | <derived2> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  2 | DERIVED     | t3         | const  | PRIMARY,idx\_t3\_id | PRIMARY | 4       |      |    1 |       |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+

### **table**

显示这一行的数据是关于哪张表的.  
有时不是真实的表名字,看到的是derivedx(x是个数字,我的理解是第几步执行的结果)  
mysql> explain select \* from (select \* from ( select \* from t3 where id=3952602) a) b;  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
| id | select\_type | table      | type   | possible\_keys     | key     | key\_len | ref  | rows | Extra |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
|  1 | PRIMARY     | <derived2> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  2 | DERIVED     | <derived3> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  3 | DERIVED     | t3         | const  | PRIMARY,idx\_t3\_id | PRIMARY | 4       |      |    1 |       |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+

### **type**

 这列很重要,显示了连接使用了哪种类别,有无使用索引.  
从最好到最差的连接类型为const、eq\_reg、ref、range、indexhe和ALL  
   
****system****

  这是const联接类型的一个特例。表仅有一行满足条件.如下(t3表上的id是primary key)  
  mysql> explain select \* from (select \* from t3 where id=3952602) a ;  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
| id | select\_type | table      | type   | possible\_keys     | key     | key\_len | ref  | rows | Extra |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+  
|  1 | PRIMARY     | <derived2> | system | NULL              | NULL    | NULL    | NULL |    1 |       |  
|  2 | DERIVED     | t3         | const  | PRIMARY,idx\_t3\_id | PRIMARY | 4       |      |    1 |       |  
+----+-------------+------------+--------+-------------------+---------+---------+------+------+-------+

****const****

表最多有一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被优化器剩余部分认为是常数。const表很快，因为它们只读取一次！  
const用于用常数值比较PRIMARY KEY或UNIQUE索引的所有部分时。在下面的查询中，tbl\_name可以用于const表：  
SELECT \* from tbl\_name WHERE primary\_key=1；  
SELECT \* from tbl\_name WHERE primary\_key\_part1=1和primary\_key\_part2=2；  
  例如:  
mysql> explain select \* from t3 where id=3952602;  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+  
| id | select\_type | table | type  | possible\_keys     | key     | key\_len | ref   | rows | Extra |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+  
|  1 | SIMPLE      | t3    | const | PRIMARY,idx\_t3\_id | PRIMARY | 4       | const |    1 |       |  
+----+-------------+-------+-------+-------------------+---------+---------+-------+------+-------+  
   
   
****eq\_ref****

对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了const类型。它用在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY。  
eq\_ref可以用于使用= 操作符比较的带索引的列。比较值可以为常量或一个使用在该表前面所读取的表的列的表达式。  
  在下面的例子中，MySQL可以使用eq\_ref联接来处理ref\_tables：  
  SELECT \* FROM ref\_table,other\_table  
  WHERE ref\_table.key\_column=other\_table.column;  
  SELECT \* FROM ref\_table,other\_table  
   WHERE ref\_table.key\_column\_part1=other\_table.column  
    AND ref\_table.key\_column\_part2=1;

  ****例如****

mysql> create unique index  idx\_t3\_id on t3(id) ;  
Query OK, 1000 rows affected (0.03 sec)  
Records: 1000  Duplicates: 0  Warnings: 0  
  mysql> explain select \* from t3,t4 where t3.id=t4.accountid;  
+----+-------------+-------+--------+-------------------+-----------+---------+----------------------+------+-------+  
| id | select\_type | table | type   | possible\_keys     | key       | key\_len | ref                  | rows | Extra |  
+----+-------------+-------+--------+-------------------+-----------+---------+----------------------+------+-------+  
|  1 | SIMPLE      | t4    | ALL    | NULL              | NULL      | NULL    | NULL                 | 1000 |       |  
|  1 | SIMPLE      | t3    | eq\_ref | PRIMARY,idx\_t3\_id | idx\_t3\_id | 4       | dbatest.t4.accountid |    1 |       |  
+----+-------------+-------+--------+-------------------+-----------+---------+----------------------+------+-------+  
  
****ref****

对于每个来自于前面的表的行组合，所有有匹配索引值的行将从这张表中读取。如果联接只使用键的最左边的前缀，或如果键不是UNIQUE或PRIMARY KEY（换句话说，如果联接不能基于关键字选择单个行的话），则使用ref。如果使用的键仅仅匹配少量行，该联接类型是不错的。  
  ref可以用于使用=或<=>操作符的带索引的列。  
  在下面的例子中，MySQL可以使用ref联接来处理ref\_tables：  
  SELECT \* FROM ref\_table WHERE key\_column=expr;  
  SELECT \* FROM ref\_table,other\_table  
   WHERE ref\_table.key\_column=other\_table.column;  
  SELECT \* FROM ref\_table,other\_table  
   WHERE ref\_table.key\_column\_part1=other\_table.column  
    AND ref\_table.key\_column\_part2=1;

****例如:****

  mysql> drop index idx\_t3\_id on t3;  
Query OK, 1000 rows affected (0.03 sec)  
Records: 1000  Duplicates: 0  Warnings: 0  
  mysql> create index idx\_t3\_id on t3(id) ;  
Query OK, 1000 rows affected (0.04 sec)  
Records: 1000  Duplicates: 0  Warnings: 0  
  mysql> explain select \* from t3,t4 where t3.id=t4.accountid;  
+----+-------------+-------+------+-------------------+-----------+---------+----------------------+------+-------+  
| id | select\_type | table | type | possible\_keys     | key       | key\_len | ref                  | rows | Extra |  
+----+-------------+-------+------+-------------------+-----------+---------+----------------------+------+-------+  
|  1 | SIMPLE      | t4    | ALL  | NULL              | NULL      | NULL    | NULL                 | 1000 |       |  
|  1 | SIMPLE      | t3    | ref  | PRIMARY,idx\_t3\_id | idx\_t3\_id | 4       | dbatest.t4.accountid |    1 |       |  
+----+-------------+-------+------+-------------------+-----------+---------+----------------------+------+-------+  
2 rows in set (0.00 sec)

****ref\_or\_null****

该联接类型如同ref，但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。  
  在下面的例子中，MySQL可以使用ref\_or\_null联接来处理ref\_tables：  
  SELECT \* FROM ref\_table  
WHERE key\_column=expr OR key\_column IS NULL;

****index\_merge****

该联接类型表示使用了索引合并优化方法。在这种情况下，key列包含了使用的索引的清单，key\_len包含了使用的索引的最长的关键元素。  
  例如:  
mysql> explain select \* from t4 where id=3952602 or accountid=31754306 ;  
+----+-------------+-------+-------------+----------------------------+----------------------------+---------+------+------+------------------------------------------------------+  
| id | select\_type | table | type        | possible\_keys              | key                        | key\_len | ref  | rows | Extra                                                |  
+----+-------------+-------+-------------+----------------------------+----------------------------+---------+------+------+------------------------------------------------------+  
|  1 | SIMPLE      | t4    | index\_merge | idx\_t4\_id,idx\_t4\_accountid | idx\_t4\_id,idx\_t4\_accountid | 4,4     | NULL |    2 | Using union(idx\_t4\_id,idx\_t4\_accountid); Using where |  
+----+-------------+-------+-------------+----------------------------+----------------------------+---------+------+------+------------------------------------------------------+  
1 row in set (0.00 sec)

****unique\_subquery****

  该类型替换了下面形式的IN子查询的ref：  
  value IN (SELECT primary\_key FROM single\_table WHERE some\_expr)  
unique\_subquery是一个索引查找函数，可以完全替换子查询，效率更高。  
   
****index\_subquery****

该联接类型类似于unique\_subquery。可以替换IN子查询，但只适合下列形式的子查询中的非唯一索引：

  value IN (SELECT key\_column FROM single\_table WHERE some\_expr)

****range****

只检索给定范围的行，使用一个索引来选择行。key列显示使用了哪个索引。key\_len包含所使用索引的最长关键元素。在该类型中ref列为NULL。  
当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN或者IN操作符，用常量比较关键字列时，可以使用range  
  mysql> explain select \* from t3 where id=3952602 or id=3952603 ;  
+----+-------------+-------+-------+-------------------+-----------+---------+------+------+-------------+  
| id | select\_type | table | type  | possible\_keys     | key       | key\_len | ref  | rows | Extra       |  
+----+-------------+-------+-------+-------------------+-----------+---------+------+------+-------------+  
|  1 | SIMPLE      | t3    | range | PRIMARY,idx\_t3\_id | idx\_t3\_id | 4       | NULL |    2 | Using where |  
+----+-------------+-------+-------+-------------------+-----------+---------+------+------+-------------+  
1 row in set (0.02 sec)

****index****

该联接类型与ALL相同，除了只有索引树被扫描。这通常比ALL快，因为索引文件通常比数据文件小。  
  当查询只使用作为单索引一部分的列时，MySQL可以使用该联接类型。

****ALL****

对于每个来自于先前的表的行组合，进行完整的表扫描。如果表是第一个没标记const的表，这通常不好，并且通常在它情况下很差。通常可以增加更多的索引而不要使用ALL，使得行能基于前面的表中的常数值或列值被检索出。

### **possible\_keys**

possible\_keys列指出MySQL能使用哪个索引在该表中找到行。注意，该列完全独立于EXPLAIN输出所示的表的次序。这意味着在possible\_keys中的某些键实际上不能按生成的表次序使用。  
如果该列是NULL，则没有相关的索引。在这种情况下，可以通过检查WHERE子句看是否它引用某些列或适合索引的列来提高你的查询性能。如果是这样，创造一个适当的索引并且再次用EXPLAIN检查查询。

### **key**

key列显示MySQL实际决定使用的键（索引）。如果没有选择索引，键是NULL。要想强制MySQL使用或忽视possible\_keys列中的索引，在查询中使用FORCE INDEX、USE INDEX或者IGNORE INDEX。

### **key\_len**

 key\_len列显示MySQL决定使用的键长度。如果键是NULL，则长度为NULL。  
使用的索引的长度。在不损失精确性的情况下，长度越短越好

### **ref**

 ref列显示使用哪个列或常数与key一起从表中选择行。

### **rows**

 rows列显示MySQL认为它执行查询时必须检查的行数。

### **Extra**

 该列包含MySQL解决查询的详细信息,下面详细.

****Distinct****

一旦MYSQL找到了与行相联合匹配的行，就不再搜索了

****Not exists****

MYSQL优化了LEFT JOIN，一旦它找到了匹配LEFT JOIN标准的行，  
  就不再搜索了

****Range checked for each****

* Record（index map:#）

             没有找到理想的索引，因此对于从前面表中来的每一个行组合，MYSQL检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一

****Using filesort****

看到这个的时候，查询就需要优化了。MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行

****Using index****

列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候

****Using temporary****

看到这个的时候，查询需要优化了。这里，MYSQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上

****Using where****

使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，并且连接类型ALL或index，这就会发生，或者是查询有问题

## **MySQL执行计划的局限**

* EXPLAIN不会告诉你关于触发器、存储过程的信息或用户自定义函数对查询的影响情况
* EXPLAIN不考虑各种Cache
* EXPLAIN不能显示MySQL在执行查询时所作的优化工作
* 部分统计信息是估算的，并非精确值
* EXPALIN只能解释SELECT操作，其他操作要重写为SELECT后查看执行计划

# **Mysql主从配置**

大型网站为了缓解大量的并发访问，除了在网站实现分布式负载均衡，远远不够。到了数据业务层、数据访问层，如果还是传统的数据结构，或者只是单单靠一台服务器扛，如此多的数据库连接操作，数据库必然会崩溃，数据丢失的话，后果更是 不堪设想。这时候，我们会考虑如何减少数据库的联接，一方面采用优秀的代码框架，进行代码的优化，采用优秀的数据缓存技术如：memcached,如果资金丰厚的话，必然会想到架设服务器群，来分担主数据库的压力。利用MySQL主从配置，实现读写分离，减轻数据库压力。这种方式，在如今很多网站里都有使用，也不是什么新鲜事情，在这边总结一下也方便大家学习参考一下。

## **概述**

搭设一台Master服务器（centos6.6系统，ip：192.168.0.101），搭设两台Slave服务器（虚拟机centos 6.6系统 ip:192.168.0.102, ip:192.168.0.103）  
****原理：****主服务器（Master）负责网站NonQuery操作，从服务器负责Query操作，用户可以根据网站功能模特性块固定访问Slave服务器，或者自己写个池或队列，自由为请求分配从服务器连接。主从服务器利用MySQL的二进制日志文件，实现数据同步。二进制日志由主服务器产生，从服务器响应获取同步数据库。

## **配置Master主服务器**

### **创建repl用户**

在Master MySQL上创建一个用户‘repl’，并允许其他Slave服务器可以通过远程访问Master，通过该用户读取二进制日志，实现数据同步。

**[sql]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. **create** user repl; //创建新用户
2. //repl用户必须具有REPLICATION SLAVE权限，除此之外没有必要添加不必要的权限，密码为mysql。说明一下192.168.0.%，这个配置是指明repl用户所在服务器，这里%是通配符，表示192.168.0.0-192.168.0.255的Server都可以以repl用户登陆主服务器。当然你也可以指定固定Ip。
3. mysql> **GRANT** REPLICATION SLAVE **ON** \*.\* **TO** 'repl'@'192.168.0.%' IDENTIFIED **BY** 'mysql';

### **开启bin-long**

找到mySQL安装文件夹修改my.Ini文件。mySQL中有好几种日志方式，这不是今天的重点。我们只要启动二进制日志log-bin就ok。  
在[mysqld]下面增加下面几行代码

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. log-bin=master-bin
2. log-bin-index=master-bin.index
3. server-id=101

重启服务后查看mySQL日志信息

****mysql> SHOW MASTER STATUS;****  
+-------------------+----------+--------------+------------------+  
| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |  
+-------------------+----------+--------------+------------------+  
| master-bin.000001 | 1285 | | |  
+-------------------+----------+--------------+------------------+  
1 row in set (0.00 sec)

## **配置Slave从服务器**

修改my.cnf文件  
关键在于下面几行

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. [mysqld]
2. init\_connect='SET autocommit=0'
3. server-id=102
4. log-bin=slave-bin
5. log-bin-index=slave-bin.index

## **在mySQL层面建立MASTER-SLAVE的连接**

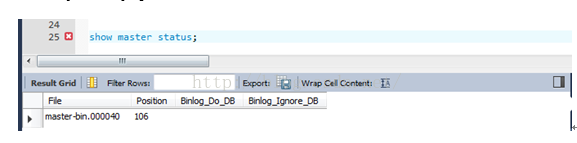
### **第一步：重启MASTER和SLAVE的两个mySQL进程**

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. service mysqld restart

### **第二步：查看master状态**

登录myaster mysql



记下这两个值

### **第三步：在slave上建立指向master的连接**

先暂停slave的mySQL Service，使用命令：

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. stop slave;

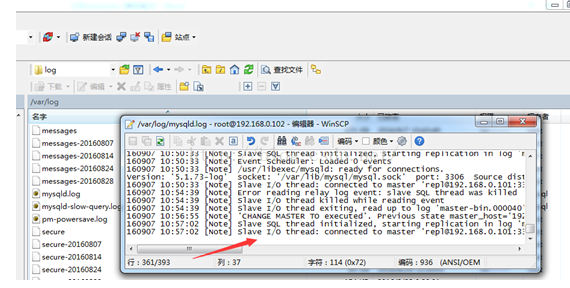


以上这步 log\_file和 master\_log\_pos必须和当前master上的show master status命令所显示的值完全一模一样，比如説我们现在加了一个slave，再加一个slave，再加第二个slave。都是如此操作。  
启动 slave的mySQL Service使用

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. start slave;

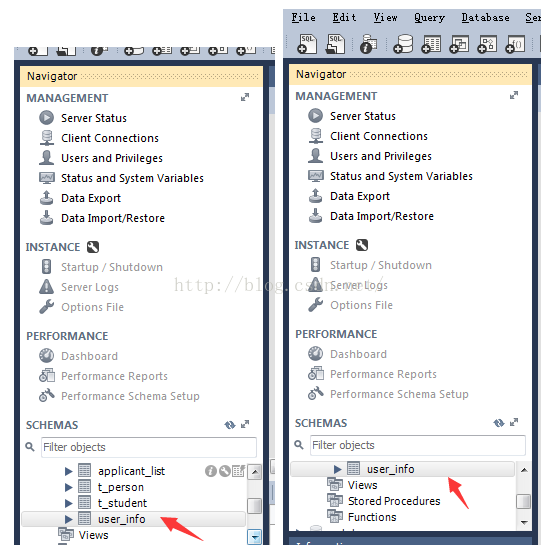
### **第四步：查看slave上的log**



## **实验master slave上的replic功能**

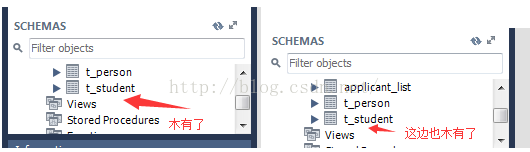
### **第一步**

在master上查看，有一张表叫user\_info。然后在slave上查看，也有一张同样的表



### **第二步**

删除master上的user\_info表，观察slave上，该表也自动被删除了。



因此，你在master上做的任何操作，都会被自动同步至slave上。

## **mysql中master宕机后，将slave升级为master的过程**

1、从slave（slave1）群众中选定一个slave，准备钱换成master；

2、检查slave1的复制状态：  
mysql> SHOW PROCESSLIST ;  
+—-+————-+———–+——+———+————+———————————————————————–+——————+  
| Id | User        | Host      | db   | Command | Time       | State                                                                 | Info             |  
+—-+————-+———–+——+———+————+———————————————————————–+——————+  
|  1 | system user |           | NULL | Connect |      54733 | Waiting for master to send event                                      | NULL             |  
|  2 | system user |           | NULL | Connect | 4294965772 |Has read all relay log; waiting for the slave I/O thread to update it | NULL             |  
|  8 | root        | localhost | NULL | Query   |          0 | NULL                                                                  | SHOW PROCESSLIST |  
+—-+————-+———–+——+———+————+———————————————————————–+——————+  
 ****注：****  
****这里主要是检测Slave1是否已经应用完从Master读取过来的在relay log中的操作，如果未应用完不能stop slave，否则数据肯定会有丢失。****

3、停止slave1的slave进程，并reset称master：  
mysql> STOP SLAVE ;  
Query OK, 0 rows affected (0.00 sec)  
mysql> RESET MASTER;  
Query OK, 0 rows affected, 8 warnings (0.02 sec)

4、将slave 群中的其他slave（slave2）的Master切换成新的master（由原slave1 reset成的）：

    1) 停slave进程：  
      mysql> stop slave;  
      Query OK, 0 rows affected (0.00 sec)  
    2) 更换master：  
      mysql> CHANGE MASTER TO  
           MASTER\_HOST=’10.0.65.106′,  
          MASTER\_USER=’repl’,  
          MASTER\_PASSWORD=’slavepass’  
          ;  
      Query OK, 0 rows affected (0.00 sec)  
    3) 开启slave：  
      mysql> start slave;  
      Query OK, 0 rows affected (0.00 sec)

5、检查slave的状态：  
mysql> show slave status;

## **没有宕机的情况下master和slave的线上热切换步骤**

1、必须使要被降级的master变为只读状态。

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/52594397" \o "copy)

1. set global read\_only=1

2、执行show master status，检查并确认已经进入只读状态。  
3、执行stop slave停止与master的同步，执行reset slave解除与master的slave关系。  
4、执行change master与另一台mysql建立slave关系。  
5、按照《实验master slave上的replic功能》小节中的内容再把被降级的master做成一个新的slave挂到new master下即可。